Syntactic ASP Forgetting with Forks

F. Aguado[1], P. Cabalar[1], J. Fandiño[2], D. Pearce[3], G. Pérez[1], C. Vidal[1]

[1] University of Corunna, Spain
[2] University of Nebraska at Omaha,USA
[3] Polytechnic University of Madrid, Spain

LPNMR 2022
September 7th, 2022, Italy

Using auxiliary atoms in ASP

- A program Π may use auxiliary atoms from $A \subseteq At$

Using auxiliary atoms in ASP

- A program Π may use auxiliary atoms from $A \subseteq At$
- Stable models of Π only use atoms in $V = At \setminus A$

Using auxiliary atoms in ASP

- A program Π may use auxiliary atoms from $A \subseteq At$
- Stable models of Π only use atoms in $V = At \setminus A$
- A forgetting operator $f(\Pi, A) = \Pi'$ in V
- Π and Π' should have the "same behaviour" on V

Using auxiliary atoms in ASP

- A program Π may use auxiliary atoms from $A \subseteq At$
- Stable models of Π only use atoms in $V = At \setminus A$
- A forgetting operator $f(\Pi, A) = \Pi'$ in V
- Π and Π' should have the "same behaviour" on V
- Strong equivalence

$$SM[\Pi \cup Q] = SM[f(\Pi, A) \cup Q] \qquad \text{for any program } Q$$

Using auxiliary atoms in ASP

- A program Π may use auxiliary atoms from $A \subseteq At$
- Stable models of Π only use atoms in $V = At \setminus A$
- A forgetting operator $f(\Pi, A) = \Pi'$ in V
- Π and Π' should have the "same behaviour" on V
- Strong ~~equivalence~~ persistence

$$SM_V[\Pi \cup Q] = SM[f(\Pi, A) \cup Q] \qquad \text{for any program } Q \text{ in } V$$

We cannot forget without forks

- [Gonçalves, Knorr & Leite 2016] forgetting under "strong persistence" is sometimes impossible in ASP. Example: a cannot be forgotten in

$$a \to c \qquad \neg a \to b \qquad \neg\neg a \to a \qquad (P_1)$$

We cannot forget without forks

- [Gonçalves, Knorr & Leite 2016] forgetting under "strong persistence" is sometimes impossible in ASP. Example: a cannot be forgotten in

$$a \to c \qquad \neg a \to b \qquad \neg\neg a \to a \qquad (P_1)$$

- Condition Ω (based on HT-models) holds iff A cannot be forgotten

We cannot forget without forks

- [Gonçalves, Knorr & Leite 2016] forgetting under "strong persistence" is sometimes impossible in ASP. Example: a cannot be forgotten in

$$a \to c \qquad \neg a \to b \qquad \neg\neg a \to a \qquad (P_1)$$

- Condition Ω (based on HT-models) holds iff A cannot be forgotten
- [Aguado *et al.* 2019] proved that forgetting under "strong persistence" is ALWAYS possible with a new operator fork '$|$'

We cannot forget without forks

- [Gonçalves, Knorr & Leite 2016] forgetting under "strong persistence" is sometimes impossible in ASP. Example: a cannot be forgotten in

$$a \to c \qquad \neg a \to b \qquad \neg\neg a \to a \qquad (P_1)$$

- Condition Ω (based on HT-models) holds iff A cannot be forgotten
- [Aguado *et al.* 2019] proved that forgetting under "strong persistence" is ALWAYS possible with a new operator fork '$|$'

$$f(P_1, a) = (b \mid c)$$

Methods for forgetting

Semantic method: obtain $f(\Pi, A)$ by generation of sets of models.
Problems:

- It always implies the worst computational cost

Methods for forgetting

Semantic method: obtain $f(\Pi, A)$ by generation of sets of models.
Problems:

- It always implies the worst computational cost
- Unfeasible by hand, even for small programs

Methods for forgetting

Semantic method: obtain $f(\Pi, A)$ by generation of sets of models.
Problems:

- It always implies the worst computational cost
- Unfeasible by hand, even for small programs
- $f(\Pi, \emptyset)$ and Π may have a (very) different syntactic look

Syntactic methods:

Methods for forgetting

Semantic method: obtain $f(\Pi, A)$ by generation of sets of models.
Problems:

- It always implies the worst computational cost
- Unfeasible by hand, even for small programs
- $f(\Pi, \emptyset)$ and Π may have a (very) different syntactic look

Syntactic methods:

- [Knorr & Leite 2014] f_{as} for non-disjunctive logic programs

Methods for forgetting

Semantic method: obtain $f(\Pi, A)$ by generation of sets of models.
Problems:

- It always implies the worst computational cost
- Unfeasible by hand, even for small programs
- $f(\Pi, \emptyset)$ and Π may have a (very) different syntactic look

Syntactic methods:

- [Knorr & Leite 2014] f_{as} for non-disjunctive logic programs
  [Berthold *et al.* 2019] f_{sp} for arbitrary logic programs
  Both satisfy strong persistence (if Ω does not hold)

Methods for forgetting

Semantic method: obtain $f(\Pi, A)$ by generation of sets of models.
Problems:

- It always implies the worst computational cost
- Unfeasible by hand, even for small programs
- $f(\Pi, \emptyset)$ and Π may have a (very) different syntactic look

Syntactic methods:

- [Knorr & Leite 2014] f_{as} for non-disjunctive logic programs
  [Berthold *et al.* 2019] f_{sp} for arbitrary logic programs
  Both satisfy strong persistence (if Ω does not hold)
- f_{sp} is rather complex: table with 10 different cases plus construction of an *as-dual* program.

In this paper

- We propose a new syntactic operator "unfolding" $f_|$ always applicable

In this paper

- We propose a new syntactic operator "unfolding" $f_|$ always applicable
- $f_|$ relies on f_c "cut operator" which is close to $\mathtt{f}_{sp}$

In this paper

- We propose a new syntactic operator "unfolding" $f_{|}$ always applicable
- $f_{|}$ relies on f_c "cut operator" which is close to $\mathtt{f}_{sp}$
- In some cases we can remove forks from the result even when $\mathtt{f}_{sp}$ is not applicable (under syntactic sufficient conditions)

Formulas and programs

We define propositional formulas φ as usual:

$$\varphi \ ::= \ \bot \ \Big\| \ p \ \Big\| \ \varphi \wedge \varphi \ \Big\| \ \varphi \vee \varphi \ \Big\| \ \varphi \rightarrow \varphi$$

Definition

An extended disjunctive rule r is an implication of the form:

$$\underbrace{p_1 \wedge \cdots \wedge p_m}_{Bd^+(r)} \wedge \underbrace{\neg p_{m+1} \wedge \cdots \wedge \neg p_n}_{Bd^-(r)} \wedge \underbrace{\neg\neg p_{n+1} \wedge \cdots \wedge \neg\neg p_k}_{Bd^{--}(r)} \rightarrow \underbrace{p_{k+1} \vee \cdots \vee p_h}_{Hd(r)}$$

where all $p_i \in At$ and $0 \leq m \leq n \leq k \leq h$.

A logic program Π is a set of extended disjunctive rules.

Formulas and programs

We define propositional formulas φ as usual:

$$\varphi \ ::= \ \bot \ \| \ p \ \| \ \varphi \wedge \varphi \ \| \ \varphi \vee \varphi \ \| \ \varphi \rightarrow \varphi$$

Definition

An extended disjunctive rule r is an implication of the form:

$$\underbrace{p_1 \wedge \cdots \wedge p_m}_{Bd^+(r)} \wedge \underbrace{\neg p_{m+1} \wedge \cdots \wedge \neg p_n}_{Bd^-(r)} \wedge \underbrace{\neg\neg p_{n+1} \wedge \cdots \wedge \neg\neg p_k}_{Bd^{--}(r)} \rightarrow \underbrace{p_{k+1} \vee \cdots \vee p_h}_{Hd(r)}$$

where all $p_i \in At$ and $0 \leq m \leq n \leq k \leq h$.
A logic program Π is a set of extended disjunctive rules.

- Any formula is strongly equivalent to a logic program [Cabalar & Ferraris 2007]

Formulas and programs

We define propositional formulas φ as usual:

$$\varphi \ ::= \ \bot \ \| \ p \ \| \ \varphi \wedge \varphi \ \| \ \varphi \vee \varphi \ \| \ \varphi \rightarrow \varphi$$

Definition

An extended disjunctive rule r is an implication of the form:

$$\underbrace{p_1 \wedge \cdots \wedge p_m}_{Bd^+(r)} \wedge \underbrace{\neg p_{m+1} \wedge \cdots \wedge \neg p_n}_{Bd^-(r)} \wedge \underbrace{\neg\neg p_{n+1} \wedge \cdots \wedge \neg\neg p_k}_{Bd^{--}(r)} \rightarrow \underbrace{p_{k+1} \vee \cdots \vee p_h}_{Hd(r)}$$

where all $p_i \in At$ and $0 \leq m \leq n \leq k \leq h$.
A logic program Π is a set of extended disjunctive rules.

- Any formula is strongly equivalent to a logic program [Cabalar & Ferraris 2007]
- A rule contains an a-choice if it has the form:

$$\varphi \wedge \neg\neg a \rightarrow \psi \vee a$$

Forks

- [Aguado *et al.* 2019] introduced a "disjunctive" operator '$|$' called fork
- $F \quad ::= \quad \bot \;\|\; p \;\|\; (F \mid F) \;\|\; F \wedge F \;\|\; \varphi \vee \varphi \;\|\; \varphi \rightarrow F$

  Keypoint: forks do not appear in disjunctions or rule bodies

$$a \wedge b \rightarrow (\neg c \rightarrow e \mid a \vee \neg d) \quad \checkmark$$

Forks

- [Aguado *et al.* 2019] introduced a "disjunctive" operator '$|$' called fork
- $F \quad ::= \quad \bot \;\|\; p \;\|\; (F \mid F) \;\|\; F \wedge F \;\|\; \varphi \vee \varphi \;\|\; \varphi \rightarrow F$

  Keypoint: forks do not appear in disjunctions or rule bodies

$$a \wedge b \rightarrow (\neg c \rightarrow e \mid a \vee \neg d) \quad \checkmark$$
$$(a \mid b) \rightarrow c \quad \times$$

Forks

- [Aguado *et al.* 2019] introduced a "disjunctive" operator '$|$' called fork
- $F \quad ::= \quad \bot \quad \| \quad p \quad \| \quad (F \mid F) \quad \| \quad F \wedge F \quad \| \quad \varphi \vee \varphi \quad \| \quad \varphi \rightarrow F$

  Keypoint: forks do not appear in disjunctions or rule bodies

$$a \wedge b \rightarrow (\neg c \rightarrow e \mid a \vee \neg d) \quad \checkmark$$
$$(a \mid b) \rightarrow c \quad \times$$

- Semantics based on Here-and-There (HT) [Heyting 1930].
  HT models = pairs $\langle H, T \rangle$ of sets of atoms $H \subseteq T$.

Forks

- [Aguado *et al.* 2019] introduced a "disjunctive" operator '$|$' called fork
- $F \quad ::= \quad \bot \;\|\; p \;\|\; (F \mid F) \;\|\; F \wedge F \;\|\; \varphi \vee \varphi \;\|\; \varphi \rightarrow F$

  Keypoint: forks do not appear in disjunctions or rule bodies

$$a \wedge b \rightarrow (\neg c \rightarrow e \mid a \vee \neg d) \quad \checkmark$$
$$(a \mid b) \rightarrow c \quad \times$$

- Semantics based on Here-and-There (HT) [Heyting 1930].
  HT models = pairs $\langle H, T \rangle$ of sets of atoms $H \subseteq T$.
- Keypoint: T-denotation of a fork F (see paper)

$$\langle\!\langle\, F \,\rangle\!\rangle^T = \text{set of sets of atoms } H\text{'s for a fixed } T \subseteq 2^{2^T}$$

Removing redundancies

Given Π, $behead^a(\Pi)$ removes:

- rules with $a \in Hd(r) \cap Bd^+(r)$

$$\varphi \wedge \cancel{a \rightarrow a} \vee \psi \quad \text{is a tautology}$$

- head occurrences of a where $a \in Hd(r) \cap Bd^-(r)$

$$\varphi \wedge \neg a \rightarrow \cancel{a} \vee \psi \;\equiv\; \varphi \wedge \neg a \rightarrow \psi$$

- $\Pi \equiv behead^a(\Pi)$

The Cut Operator

Cut inference rule

$$\frac{\varphi \wedge a \rightarrow \psi \qquad \alpha \rightarrow a \vee \beta}{} \quad \text{(CUT)}$$

The Cut Operator

Cut inference rule

$$\frac{\varphi \wedge a \rightarrow \psi \qquad \alpha \rightarrow a \vee \beta}{\varphi \wedge \alpha \rightarrow \psi \vee \beta} \quad \text{(CUT)}$$

The Cut Operator

Cut inference rule

$$\frac{\varphi \wedge a \rightarrow \psi \qquad \alpha \rightarrow a \vee \beta}{\varphi \wedge \alpha \rightarrow \psi \vee \beta} \quad \text{(CUT)}$$

Example (Π_1)

$$a \rightarrow t \qquad s \rightarrow a \qquad r \rightarrow a \vee u \qquad \neg a \rightarrow v$$

The Cut Operator

Cut inference rule

$$\frac{\varphi \wedge a \rightarrow \psi \qquad \alpha \rightarrow a \vee \beta}{\varphi \wedge \alpha \rightarrow \psi \vee \beta} \text{(CUT)}$$

Example (Π_1)

$$a \rightarrow t \qquad s \rightarrow a \qquad r \rightarrow a \vee u \qquad \neg a \rightarrow v$$

$$\frac{a \rightarrow t \qquad s \rightarrow a}{}$$

$$\frac{a \rightarrow t \qquad r \rightarrow a \vee u}{}$$

The Cut Operator

Cut inference rule

$$\frac{\varphi \wedge a \rightarrow \psi \qquad \alpha \rightarrow a \vee \beta}{\varphi \wedge \alpha \rightarrow \psi \vee \beta} \text{(CUT)}$$

Example (Π_1)

$$a \rightarrow t \qquad s \rightarrow a \qquad r \rightarrow a \vee u \qquad \neg a \rightarrow v$$

$$\frac{a \rightarrow t \qquad s \rightarrow a}{s \rightarrow t} \qquad \frac{a \rightarrow t \qquad r \rightarrow a \vee u}{r \rightarrow t \vee u}$$

The Cut Operator

- $NES(\Pi, a, r)$ = conjunction of all applications of cut for a on rule r ($a \in B^+(r)$)

The Cut Operator

- $NES(\Pi, a, r)$ = conjunction of all applications of cut for a on rule r ($a \in B^+(r)$)

Example (Π_1)

$$a \rightarrow t \qquad s \rightarrow a \qquad r \rightarrow a \vee u \qquad \neg a \rightarrow v$$

The Cut Operator

- $NES(\Pi, a, r)$ = conjunction of all applications of cut for a on rule r ($a \in B^+(r)$)

Example (Π_1)

$$a \to t \qquad s \to a \qquad r \to a \vee u \qquad \neg a \to v$$

$$NES(\Pi_1, a, (a \to t)) \;=\; (s \to t) \wedge (r \to t \vee u)$$

The Cut Operator

- $NES(\Pi, a, r)$ = conjunction of all applications of cut for a on rule r ($a \in B^+(r)$)

Example (Π_1)

$$a \to t \qquad s \to a \qquad r \to a \vee u \qquad \neg a \to v$$

$$NES(\Pi_1, a, (a \to t)) \;=\; (s \to t) \wedge (r \to t \vee u)$$

- Special case, when the rule is $\neg a = (a \wedge \top \to \bot)$
  $NES(\Pi, a, \neg a)$ = conjunction of rules with $a \in Hd(r)$ replacing a by $\bot$

The Cut Operator

- $NES(\Pi, a, r)$ = conjunction of all applications of cut for a on rule r ($a \in B^+(r)$)

Example (Π_1)

$$a \to t \qquad s \to a \qquad r \to a \vee u \qquad \neg a \to v$$

$$NES(\Pi_1, a, (a \to t)) \;=\; (s \to t) \wedge (r \to t \vee u)$$

- Special case, when the rule is $\neg a = (a \wedge \top \to \bot)$
  $NES(\Pi, a, \neg a)$ = conjunction of rules with $a \in Hd(r)$ replacing a by $\bot$

- In the example:

$$NES(\Pi_1, a, \neg a) = (s \to \bot) \wedge (r \to \bot \vee u)$$

The Cut Operator

- $NES(\Pi, a, r)$ = conjunction of all applications of cut for a on rule r ($a \in B^+(r)$)

Example (Π_1)

$$a \to t \qquad s \to a \qquad r \to a \vee u \qquad \neg a \to v$$

$$NES(\Pi_1, a, (a \to t)) \;=\; (s \to t) \wedge (r \to t \vee u)$$

- Special case, when the rule is $\neg a = (a \wedge \top \to \bot)$
  $NES(\Pi, a, \neg a)$ = conjunction of rules with $a \in Hd(r)$ replacing a by $\bot$

- In the example:

$$NES(\Pi_1, a, \neg a) = (s \to \bot) \wedge (r \to \bot \vee u) \equiv \neg s \wedge (r \to u)$$

The Cut Operator $\mathtt{f}_c(\Pi, a)$

(i) Remove atom a from non-supporting heads obtaining $\Pi' = behead^a(\Pi)$;

The Cut Operator $\mathtt{f}_c(\Pi, a)$

(i) Remove atom a from non-supporting heads obtaining $\Pi' = \mathit{behead}^a(\Pi)$;

Example (Π_1)

$$a \rightarrow t \qquad s \rightarrow a \qquad r \rightarrow a \vee u \qquad \neg a \rightarrow v$$

In the example: $\Pi_1 = \mathit{behead}^a(\Pi_1)$ so nothing to do

The Cut Operator $\mathtt{f}_c(\Pi, a)$

(ii) Replace each rule $r \in \Pi'$ with $a \in B^+(r)$ by $NES(\Pi', a, r)$

Example

$$a \rightarrow t \qquad s \rightarrow a \qquad r \rightarrow a \vee u \qquad \neg a \rightarrow v$$

The Cut Operator $\mathtt{f}_c(\Pi, a)$

(ii) Replace each rule $r \in \Pi'$ with $a \in B^+(r)$ by $NES(\Pi', a, r)$

Example

$$a \to t \qquad s \to a \qquad r \to a \vee u \qquad \neg a \to v$$

$$NES(\Pi_1, a, (a \to t)) \;=\; (s \to t) \wedge (r \to t \vee u)$$

The Cut Operator $\mathtt{f}_c(\Pi, a)$

(ii) Replace each rule $r \in \Pi'$ with $a \in B^+(r)$ by $NES(\Pi', a, r)$

Example

$$(s \to t) \wedge (r \to t \vee u) \qquad s \to a \qquad r \to a \vee u \qquad \neg a \to v$$

$$NES(\Pi_1, a, (a \to t)) \;=\; (s \to t) \wedge (r \to t \vee u)$$

The Cut Operator $\mathtt{f}_c(\Pi, a)$

(iii) Remove every rule r with $Hd(r) = \{a\}$;

Example

$$(s \to t) \wedge (r \to t \vee u) \qquad s \to a \qquad r \to a \vee u \qquad \neg a \to v$$

The Cut Operator $\mathtt{f}_c(\Pi, a)$

(iii) Remove every rule r with $Hd(r) = \{a\}$;

Example

$$(s \to t) \wedge (r \to t \vee u) \qquad \cancel{s \to a} \qquad r \to a \vee u \qquad \neg a \to v$$

The Cut Operator $\mathtt{f}_c(\Pi, a)$

(iii) Remove every rule r with $Hd(r) = \{a\}$;

Example

$$(s \rightarrow t) \wedge (r \rightarrow t \vee u) \qquad r \rightarrow a \vee u \qquad \neg a \rightarrow v$$

The Cut Operator $\mathtt{f}_c(\Pi, a)$

(iv) Replace the remaining occurrences of a by $\neg NES(\Pi', a, \neg a)$.

Example

$$(s \rightarrow t) \wedge (r \rightarrow t \vee u) \quad r \rightarrow a \vee u \quad \neg a \rightarrow v$$

The Cut Operator $\mathtt{f}_c(\Pi, a)$

(iv) Replace the remaining occurrences of a by $\neg NES(\Pi', a, \neg a)$.

Example

$$(s \to t) \wedge (r \to t \vee u) \quad r \to a \vee u \quad \neg a \to v$$

$$NES(\Pi_1, a, \neg a) = \neg s \wedge (r \to u)$$
$$a \rightsquigarrow \neg(\neg s \wedge (r \to u))$$
$$\neg a \rightsquigarrow \neg\neg(\neg s \wedge (r \to u))$$

The Cut Operator $\mathtt{f}_c(\Pi, a)$

(iv) Replace the remaining occurrences of a by $\neg NES(\Pi', a, \neg a)$.

Example

$$(s \to t) \wedge (r \to t \vee u) \quad r \to \neg(\neg s \wedge (r \to u)) \vee u \quad \neg\neg(\neg s \wedge (r \to u)) \to v$$

$$NES(\Pi_1, a, \neg a) = \neg s \wedge (r \to u)$$
$$a \quad \rightsquigarrow \quad \neg(\neg s \wedge (r \to u))$$
$$\neg a \quad \rightsquigarrow \quad \neg\neg(\neg s \wedge (r \to u))$$

The Cut Operator $\mathtt{f}_c(\Pi, a)$

(iv) Replace the remaining occurrences of a by $\neg NES(\Pi', a, \neg a)$.

Example

$$(s \to t) \wedge (r \to t \vee u) \quad r \to \neg(\neg s \wedge (r \to u)) \vee u \quad \neg\neg(\neg s \wedge (r \to u)) \to v$$

$$\begin{aligned} NES(\Pi_1, a, \neg a) &= \neg s \wedge (r \to u) \\ a &\rightsquigarrow \neg(\neg s \wedge (r \to u)) \\ \neg a &\rightsquigarrow \neg\neg(\neg s \wedge (r \to u)) \end{aligned}$$

The result is a set of formulas, but they can be easily reduced to a program by well-known HT transformations [Cabalar et al 2005]

The Cut Operator $\mathtt{f}_c(\Pi, a)$

(iv) Replace the remaining occurrences of a by $\neg NES(\Pi', a, \neg a)$.

Example

$$(s \to t) \wedge (r \to t \vee u) \quad r \to \neg(\neg s \wedge (r \to u)) \vee u \quad \neg\neg(\neg s \wedge (r \to u)) \to v$$

$$NES(\Pi_1, a, \neg a) = \neg s \wedge (r \to u)$$
$$a \quad \rightsquigarrow \quad \neg(\neg s \wedge (r \to u))$$
$$\neg a \quad \rightsquigarrow \quad \neg\neg(\neg s \wedge (r \to u))$$

The result is a set of formulas, but they can be easily reduced to a program by well-known HT transformations [Cabalar et al 2005]

Example

$$s \to t \qquad r \to t \vee u \qquad \neg s \wedge \neg r \to v$$
$$\neg s \wedge \neg\neg u \to v \qquad r \wedge \neg s \wedge \neg\neg u \to u$$

When does $\mathtt{f}_c$ guarantee strong persistence?

Definition (Definition 4 from [Berthold *et al.* 2019])

Π is a-forgettable if, at least one of the following conditions is satisfied:

1. Π contains the fact 'a' as a rule

$$(\top \rightarrow a) \in \Pi$$

When does $\mathtt{f}_c$ guarantee strong persistence?

Definition (Definition 4 from [Berthold *et al.* 2019])

Π is a-forgettable if, at least one of the following conditions is satisfied:

1. Π contains the fact 'a' as a rule

$$(\top \rightarrow a) \in \Pi$$

2. Π does not contain a-choices

$$(\varphi \wedge \neg\neg a \rightarrow a \vee \psi) \notin \Pi$$

When does $\mathtt{f}_c$ guarantee strong persistence?

Definition (Definition 4 from [Berthold *et al.* 2019])

Π is a-forgettable if, at least one of the following conditions is satisfied:

1. Π contains the fact 'a' as a rule

$$(\top \rightarrow a) \in \Pi$$

2. Π does not contain a-choices

$$(\varphi \wedge \neg\neg a \rightarrow a \vee \psi) \notin \Pi$$

3. All rules in Π in which a occurs are a-choices

$$\Pi = \{r \mid a \notin At(r)\} \cup \{\varphi \wedge \neg\neg a \rightarrow a \vee \psi\}$$

When does $\mathtt{f}_c$ guarantee strong persistence?

Theorem

Let Π be a logic program for signature At, let $V \subseteq At$ and $a \in At \setminus V$ and let $\Pi' = behead^a(\Pi)$. If Π' is a-forgettable, then:

$$\Pi \cong_V \mathtt{f}_c(\Pi, a).$$

When does $\mathtt{f}_c$ guarantee strong persistence?

Theorem

Let Π be a logic program for signature At, let $V \subseteq At$ and $a \in At \setminus V$ and let $\Pi' = behead^a(\Pi)$. If Π' is a-forgettable, then:

$$\Pi \cong_V \mathtt{f}_c(\Pi, a).$$

Example (Π_1 has no a-choices)

$$a \to t \qquad s \to a \qquad \neg a \to v \qquad r \to a \vee u$$

$\mathtt{f}_c(\Pi_1, a)$:

$$s \to t \quad r \to t \vee u \quad \neg s \wedge \neg r \to v \quad \neg s \wedge \neg\neg u \to v \quad r \wedge \neg s \wedge \neg\neg u \to u$$

Forgetting into forks: the unfolding operator

Example (P_1)

$$a \to b \qquad \neg a \to c \qquad \neg\neg a \to a$$

P_1 contains an a-choice.

$\mathtt{f}_c$ does not guarantee strong persistence

Forgetting into forks: the unfolding operator

- $\Pi[\gamma/\varphi]$: substitute γ by φ
- Propositional logic: forgetting a is equivalent to $\Pi[a/\bot] \vee \Pi[a/\top]$

Forgetting into forks: the unfolding operator

- $\Pi[\gamma/\varphi]$: substitute γ by φ
- Propositional logic: forgetting a is equivalent to $\Pi[a/\bot] \vee \Pi[a/\top]$
- Equilibrium logic: $\Pi \equiv (\Pi \wedge \neg a \mid \Pi \wedge \neg\neg a)$

Forgetting into forks: the unfolding operator

- $\Pi[\gamma/\varphi]$: substitute γ by φ
- Propositional logic: forgetting a is equivalent to $\Pi[a/\bot] \vee \Pi[a/\top]$
- Equilibrium logic: $\Pi \equiv (\Pi \wedge \neg a \mid \Pi \wedge \neg\neg a)$
- $\Pi \wedge \neg a$ and $\Pi \wedge \neg\neg a$ are a-forgettable

$$\Pi \wedge \neg a \equiv \Pi[a/\bot] \wedge \neg a$$

$$\neg\neg a \wedge \varphi \rightarrow a \vee \psi \equiv \bot \wedge \varphi \rightarrow \bot \vee \psi \equiv \top$$

Forgetting into forks: the unfolding operator

- $\Pi[\gamma/\varphi]$: substitute γ by φ
- Propositional logic: forgetting a is equivalent to $\Pi[a/\bot] \vee \Pi[a/\top]$
- Equilibrium logic: $\Pi \equiv (\Pi \wedge \neg a \mid \Pi \wedge \neg\neg a)$
- $\Pi \wedge \neg a$ and $\Pi \wedge \neg\neg a$ are a-forgettable

$$\Pi \wedge \neg\neg a \equiv \Pi[\neg a/\bot] \wedge \neg\neg a$$

$$\neg\neg a \wedge \varphi \rightarrow a \vee \psi \equiv \top \wedge \varphi \rightarrow a \vee \psi \equiv \varphi \rightarrow a \vee \psi$$

Forgetting into forks: the unfolding operator

- $\Pi \equiv (\Pi \wedge \neg a \mid \Pi \wedge \neg\neg a)$
- $\Pi \wedge \neg a$ and $\Pi \wedge \neg\neg a$ are a-forgettable

Definition (Unfolding operator, $\mathtt{f}_{|}$)

For any logic program Π and atom a we define:

$$\mathtt{f}_{|}(\Pi, a) \stackrel{\text{def}}{=} (\ \mathtt{f}_c(\Pi \wedge \neg a, a)\ \mid\ \mathtt{f}_c(\Pi \wedge \neg\neg a, a)\ )$$

Forgetting into forks: the unfolding operator

- $\Pi \equiv (\Pi \wedge \neg a \mid \Pi \wedge \neg\neg a)$
- $\Pi \wedge \neg a$ and $\Pi \wedge \neg\neg a$ are a-forgettable

Definition (Unfolding operator, $\mathtt{f}_{|}$)

For any logic program Π and atom a we define:

$$\mathtt{f}_{|}(\Pi, a) \stackrel{\text{def}}{=} (\ \mathtt{f}_c(\Pi \wedge \neg a, a)\ \mid\ \mathtt{f}_c(\Pi \wedge \neg\neg a, a)\ )$$

Theorem

Let Π be a logic program for signature At, $V \subseteq At$ and $a \in At \setminus V$. Then,

$$\Pi \cong_V \mathtt{f}_{|}(\Pi, a).$$

The unfolding operator: an example

Example (P_1)

$$a \to c \qquad \neg a \to b \qquad \neg\neg a \to a \qquad (P_1)$$

$$\mathtt{f}_{|}(P_1, a) = (\ \mathtt{f}_c(P_1 \wedge \neg a, a)\ \mid\ \mathtt{f}_c(P_1 \wedge \neg\neg a, a)\ )$$

The unfolding operator: an example

$$\mathtt{f}_{|}(P_1, a) = (\ \mathtt{f}_c(P_1 \wedge \neg a, a)\ \mid\ \mathtt{f}_c(P_1 \wedge \neg\neg a, a)\ )$$

Example ($\mathtt{f}_c(P_1 \wedge \neg a, a)$)

$$a \to c \qquad \neg a \to b \qquad \neg\neg a \to a \qquad \neg a \qquad (P_1 \wedge \neg a)$$

$$P_1 \wedge \neg a \ \equiv\ P_1[a/\bot] \wedge \neg a$$

The unfolding operator: an example

$$\mathtt{f}_{|}(P_1, a) = (\ \mathtt{f}_c(P_1 \wedge \neg a, a)\ \mid\ \mathtt{f}_c(P_1 \wedge \neg\neg a, a)\ )$$

Example ($\mathtt{f}_c(P_1 \wedge \neg a, a)$)

$$a \to c \qquad \neg a \to b \qquad \neg\neg a \to a \qquad \neg a \qquad (P_1 \wedge \neg a)$$

$$\bot \to c \qquad \neg\bot \to b \qquad \neg\neg\bot \to \bot \qquad \neg\bot \qquad (P_1 \wedge \neg a)$$

$$P_1 \wedge \neg a \ \equiv\ P_1[a/\bot] \wedge \neg a$$

The unfolding operator: an example

$$\mathtt{f}_{|}(P_1, a) = (\ \mathtt{f}_c(P_1 \wedge \neg a, a)\ \mid\ \mathtt{f}_c(P_1 \wedge \neg\neg a, a)\ )$$

Example ($\mathtt{f}_c(P_1 \wedge \neg a, a)$)

$a \rightarrow c$	$\neg a \rightarrow b$	$\neg\neg a \rightarrow a$	$\neg a$	$(P_1 \wedge \neg a)$
$\bot \rightarrow c$	$\neg \bot \rightarrow b$	$\neg\neg \bot \rightarrow \bot$	$\neg \bot$	$(P_1 \wedge \neg a)$
~~$\bot \rightarrow c$~~	b	~~$\neg\neg \bot \rightarrow \bot$~~	~~$\neg \bot$~~	$(P_1 \wedge \neg a)$

$$P_1 \wedge \neg a \ \equiv\ P_1[a/\bot] \wedge \neg a$$

$$\mathtt{f}_c(P_1 \wedge \neg a, a) \ \equiv\ P_1[a/\bot] = b$$

The unfolding operator: an example

$$\mathtt{f}_{|}(P_1, a) = (\ \mathtt{f}_c(P_1 \wedge \neg a, a)\ \mid\ \mathtt{f}_c(P_1 \wedge \neg\neg a, a)\ )$$

Example ($\mathtt{f}_c(P_1 \wedge \neg\neg a, a)$)

$$a \rightarrow c \qquad \neg a \rightarrow b \qquad \neg\neg a \rightarrow a \qquad \neg\neg a \qquad (P_1 \wedge \neg\neg a)$$

$$P_1 \wedge \neg\neg a \ \equiv\ P_1[\neg a/\bot] \wedge \neg\neg a$$

The unfolding operator: an example

$$\mathtt{f}_{|}(P_1, a) = (\ \mathtt{f}_c(P_1 \wedge \neg a, a)\ \mid\ \mathtt{f}_c(P_1 \wedge \neg\neg a, a)\ )$$

Example ($\mathtt{f}_c(P_1 \wedge \neg\neg a, a)$)

$$a \rightarrow c \qquad \neg a \rightarrow b \qquad \neg\neg a \rightarrow a \qquad \neg\neg a \qquad (P_1 \wedge \neg\neg a)$$

$$a \rightarrow c \qquad \bot \rightarrow b \qquad \neg\bot \rightarrow a \qquad \neg\bot \qquad (P_1 \wedge \neg\neg a)$$

$$P_1 \wedge \neg\neg a \ \equiv\ P_1[\neg a/\bot] \wedge \neg\neg a$$

The unfolding operator: an example

$$\mathtt{f}_{|}(P_1, a) = (\ \mathtt{f}_c(P_1 \wedge \neg a, a)\ \mid\ \mathtt{f}_c(P_1 \wedge \neg\neg a, a)\ )$$

Example ($\mathtt{f}_c(P_1 \wedge \neg\neg a, a)$)

$$a \rightarrow c \qquad \neg a \rightarrow b \qquad \neg\neg a \rightarrow a \qquad \neg\neg a \qquad (P_1 \wedge \neg\neg a)$$

$$a \rightarrow c \qquad \cancel{\bot \rightarrow b} \qquad a \qquad \cancel{\neg\bot} \qquad (P_1 \wedge \neg\neg a)$$

$$a \wedge c$$

$$P_1 \wedge \neg\neg a \ \equiv\ P_1[\neg a/\bot] \wedge \neg\neg a$$

$$\mathtt{f}_c(P_1 \wedge \neg\neg a, a) = \mathtt{f}_c(a \wedge c, a) = c$$

The unfolding operator: an example

Example (P_1)

$$a \to b \qquad \neg a \to c \qquad \neg\neg a \to a$$

$$\mathtt{f}_c(P_1 \wedge \neg a, a) = b \qquad \mathtt{f}_c(P_1 \wedge \neg\neg a, a) = c$$

$$\mathtt{f}_{|}(P_1, a) = (\ \mathtt{f}_c(P_1 \wedge \neg a, a)\ \mid\ \mathtt{f}_c(P_1 \wedge \neg\neg a, a)\ ) = (b\ \mid\ c)$$

Sometimes a-forgettable is not necessary

Example (P_2)

$$a \to c \qquad \neg a \to b \qquad \neg\neg a \to a \qquad \underbrace{b \to c \qquad c \to b}_{X} \qquad (P_2 = P_1 \wedge X)$$

Sometimes a-forgettable is not necessary

Example (P_2)

$$a \to c \qquad \neg a \to b \qquad \neg\neg a \to a \qquad \underbrace{b \to c \qquad c \to b}_{X} \qquad (P_2 = P_1 \wedge X)$$

$$\begin{aligned} \mathtt{f}_{|}(P_2, a) &= \mathtt{f}_{|}(P_1 \wedge X, a) \\ &= \mathtt{f}_{|}(P_1, a) \wedge X \end{aligned} \qquad \text{strong invariance}$$

Sometimes a-forgettable is not necessary

Example (P_2)

$$a \to c \qquad \neg a \to b \qquad \neg\neg a \to a \qquad \underbrace{b \to c \qquad c \to b}_{X} \qquad (P_2 = P_1 \wedge X)$$

$$\begin{aligned}
\mathtt{f}_{|}(P_2, a) &= \mathtt{f}_{|}(P_1 \wedge X, a) \\
&= \mathtt{f}_{|}(P_1, a) \wedge X \qquad \text{strong invariance} \\
&= (b \mid c) \wedge (b \to c) \wedge (c \to b) \quad (F \mid G) \wedge H \equiv (F \wedge H \mid G \wedge H) \\
&\equiv (b \wedge (b \to c) \wedge (c \to b) \mid c \wedge (b \to c) \wedge (c \to b))
\end{aligned}$$

Sometimes a-forgettable is not necessary

Example (P_2)

$$a \to c \qquad \neg a \to b \qquad \neg\neg a \to a \qquad \underbrace{b \to c \qquad c \to b}_{X} \qquad (P_2 = P_1 \wedge X)$$

$$\begin{aligned}
\mathtt{f}_{|}(P_2, a) &= \mathtt{f}_{|}(P_1 \wedge X, a) \\
&= \mathtt{f}_{|}(P_1, a) \wedge X \qquad \text{strong invariance} \\
&= (b \mid c) \wedge (b \to c) \wedge (c \to b) \quad (F \mid G) \wedge H \equiv (F \wedge H \mid G \wedge H) \\
&= (b \wedge (b \to c) \wedge (c \to b) \mid c \wedge (b \to c) \wedge (c \to b)) \\
&\equiv (b \wedge c \mid c \wedge b) \\
&\equiv b \wedge c
\end{aligned}$$

Conclusions

- We have introduced a syntactic transformation $\texttt{f}_{|}$, we called unfolding
- $\texttt{f}_{|}$ is always applicable (under strong persistence)

Conclusions

- We have introduced a syntactic transformation $\mathtt{f}_{|}$, we called unfolding
- $\mathtt{f}_{|}$ is always applicable (under strong persistence)
- $\mathtt{f}_{|}$ relies on $\mathtt{f}_{c}$ that can be applied on any program that does not contain choice rules and it returns a propositional formula without forks.
- With general properties of forks we can sometimes reduce forks to propositional formulas

Future work

- We intend to extend the syntactic conditions under which forks can be reduced to formulas

Future work

- We intend to extend the syntactic conditions under which forks can be reduced to formulas
- We will also study the extension of the unfolding operator to sets of atoms

Syntactic ASP Forgetting with Forks

F. Aguado, P. Cabalar, J. Fandiño, D. Pearce, G. Pérez, C. Vidal

Thanks for your attention!